PCT/EP 00/02276

#3

Europäisches Patentamt	European Patent Office	Office européen des brevets

09/700373

EPO - DG 1

19. 05. 2000

(86)

EP 00/2276

4

Bescheinigung Certificate Attestation

Die angehefteten Unterlagen stimmen mit der ursprünglich eingereichten Fassung der auf dem nächsten Blatt bezeichneten europäischen Patentanmeldung überein.

The attached documents are exact copies of the European patent application described on the following page, as originally filed.

Les documents fixés à cette attestation sont conformes à la version initialement déposée de la demande de brevet européen spécifiée à la page suivante.

Patentanmeldung Nr. Patent application No. Demande de brevet n°

99200776.5

PRIORITY DOCUMENT
SUBMITTED OR TRANSMITTED IN COMPLIANCE WITH RULE 17.1(a) OR (b)

Der Präsident des Europäischen Patentamts;
Im Auftrag

For the President of the European Patent Office

Le Président de l'Office européen des brevets
p.o.

Hatten

I.L.C. HATTEN-HECKMAN

DEN HAAG, DEN
THE HAGUE, 04/04/00
LA HAYE, LE

EPA/EPO/OEB Form 1014 - 02.91

Europäisches Patentamt | European Patent Office | Office européen des brevets

Blatt 2 der Bescheinigung
Sheet 2 of the certificate
Page 2 de l'attestation

Anmeldung Nr.:
Application no.: 99200776.5
Demande n°:

Anmeldetag:
Date of filing: 15/03/99
Date de dépôt:

Anmelder:
Applicant(s):
Demandeur(s):
Koninklijke Philips Electronics N.V.
5621 BA Eindhoven
NETHERLANDS

Bezeichnung der Erfindung:
Title of the invention:
Titre de l'invention:
Copy protection for SSA with removable memory modules

REC'D 30 MAY [illegible]
WIPO

In Anspruch genommene Prioriät(en) / Priority(ies) claimed / Priorité(s) revendiquée(s)

Staat: State: Pays:	Tag: Date: Date:	Aktenzeichen: File no. Numéro de dépôt:

Internationale Patentklassifikation:
International Patent classification:
Classification internationale des brevets:
/

Am Anmeldetag benannte Vertragstaaten:
Contracting states designated at date of filing: AT/BE/CH/CY/DE/DK/ES/FI/FR/GB/GR/IE/IT/LI/LU/MC/NL/PT/SE
Etats contractants désignés lors du depôt:

Bemerkungen:
Remarks:
Remarques:

Copy protection for SSA with removable memory modules

Within a few years, Solid State Audio (SSA) players are expected to become the new standard for portable audio playback devices. This is mainly due to many advantages on weight, size, power use, and last but not least, shock resistance with respect to current solutions using disc or tape. Currently available SSA players combine 32–64 MB of flash memory and audio compression techniques such as MPEG 1 layer III (MP3) or AAC to achieve up to one hour of (near) CD quality music playing time. Due to the digital nature of these devices, however, the music industry demands proper copyright protection features.

One of the tools for copy protection of digital content is encryption. While encryption by itself does not prevent illegal copying, it does render such copies useless, as the original content can be retrieved only by decrypting it using the proper key. As a result, the play back of the content is limited to those devices that have access to that key. It is a task of the copy protection system to manage the keys in such a way that illegal copying is not possible, while at the same time not inconveniencing legal and intended use of the content.

In the following, it is assumed that a SSA player employs detachable memory modules, which can be accessed by other means as well (e.g. PC based readers). Basically, two approaches exist for copy protection. The first is to bind the audio to a specific player by providing each individual player with a unique, secret, number that is used as the key for encryption of the audio. Therefore, the audio stored on memory modules by one player will play on that player only. Of course, this is very annoying if one has multiple SSA players. It is required that one is able to play music stored on a memory module, regardless of the SSA device used to download it onto the module. What should be prevented is that a user can not make copy the audio content to another module and be able to play from both.

One solution is to embed a unique identification code in the memory module, which can be read by the application, but which can not be changed. This identification code can then be used to generate an encryption key, which is specific for the module.

Another solution is to make use of natural defects in the memory (e.g. the Mostly Good Memory (MGM) approach of Hitachi to fabricate cheap but high storage capacity flash memories). The locations of these defects probably will be unique for each module, and as such can act as a 'fingerprint' of that device. Again, a unique key can be generated, which is specific for the module.

Below, a third solution is outlined, which does not require that each module is personalised using a unique identification code. At some point in time, this may be advantageous because of privacy considerations.

Most of the memory modules currently being proposed for solid state multimedia storage applications consist of a large flash memory and an on-board controller. The controller may or may not be integrated, and multiple separate memory chips may be employed on the module. Examples of multimedia memory modules are: Memory Stick (Sony), SmartMedia (SSFDC Forum), Miniature Card (MC Forum), Compact Flash (PCMCIA Forum), Multimedia Card (Sandisk, Siemens). In addition, these devices can be thought of as block devices, similar to hard disk drives, in that access occurs by addressing sectors (typically 512 bytes) on the module. Indeed, some of the modules listed above employ the ATA interface standard, which is used to connect hard disks and other peripherals to a PC. This enables easy duplication (bit by bit) of the content of such memory modules using a PC. Other modules use a proprietary interface and command set, but still are block based, i.e. individual sectors on the module can be address and modified.

To effectively use a storage device, it is necessary to implement a file system by means of which the user data is organised and accessed. By treating the memory module as a block device, the creation and management of a file system is left to the application. In a PC environment, where the operating system already has built-in file system support, this is a logical choice: by supporting the ATA standard this support can be reused for the memory module without any modification. However, in stand-alone devices, such as a SSA player, the application is burdened with file system details, if the memory module employs the block device approach. Therefore, stand-alone (portable) applications which require storage of multimedia content, may be built more efficiently if the controller on the memory module takes care of the file system details.

Without going in too much detail, an Application Programming Interface (API) is proposed for the memory module, which hides the details of the file system management, and at the same time provides features which can be employed for copy protection. The memory module is schematically represented in figure 1.

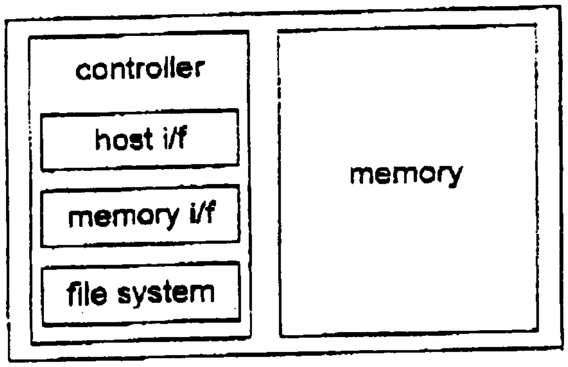

Figure 1. Schematic diagram of the memory module

The API should contain the following functionality as a minimum:

1. Format memory
Input: none
Output: volume number (optional)
Remarks: the volume number can be either a unique, hard-wired identification tag of the memory module, or a different random number which is generated each time this command is executed. It is not possible to change this number, other than by executing this command again (thereby destroying all data on the device). (The random number generator can be seeded by e.g. the fingerprint of a MGM, or thermal or electrical fluctuations in the device.) Note that the copy protection system described below, does not require a volume number (though it may be useful to have one anyway).
2. Create file
Input: none
Output: file ID
Remarks: the file ID is a number by means of which the file is referenced in all file operations. It can be reused.
3. Block write
Input: file ID, data block (e.g. a sector of 512 bytes)
Output: sector number for the next block write
Remarks: the sector number for the next block write is randomly chosen (by the controller) from the free block list. The application is free to use or discard this data at will. Note that the sectors in which the file data are stored can be chosen at random from the available sectors because the flash memory is not hampered by a seek time common in disk based systems. In addition, it helps to level wear over the entire device. Below, it is explained how this feature can be used to implement an effective copy protection scheme.
4. Close file
Input: file ID
Output: none
5. Get table of contents
Input: none
Output: table of contents
6. Open file
Input: file ID
Output: none
7. Block read
Input: file ID
Output: data block (e.g. a sector of 512 bytes), sector number for next block read
Remarks: The application is free to use or discard the sector number for next block read at will. Below it is explained how this feature can be used to implement an effective copy protection scheme.
8. Delete file
Input: file ID
Output: none

It may be clear that many other useful functions can be defined. For example, error signalling and reporting has not been taken into account. However, the above minimum functionality is used to implement an effective copy protection system. However, before going into the details of this system, a possible file format is shown in figure 2. Here, the file is split in blocks which have the size of a single sector on the memory module. The first block of the file carries information on the file, while the other blocks contain the actual data. The reason for the different use of the first block in the file will become clear later.

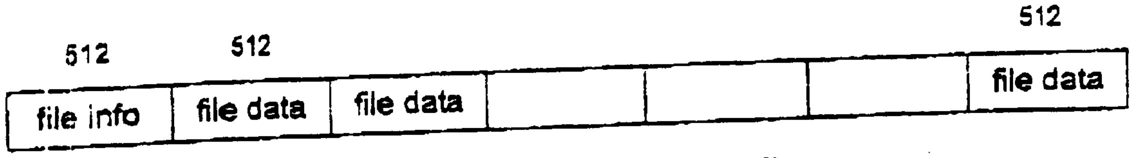

Figure 2. Example structure of a file

First it, it may be clear that the above system inherently renders it impossible to make exact bit copies of the module contents, because no means are provided for an application to modify individual sectors. First copying the contents to an intermediate storage medium and subsequently copying it to a second module will result in an exact copy of all data bits; however, the locations at which these bits are stored are widely different in both modules! This can be exploited to devise a copy protection system, as follows.

The application encrypts each data block using a key which is derived from a (globally shared) secret, and the sector number where this data block is stored. The latter information is obtained from the block write function (which returns the sector number of the next sector in the file). As this information is not available for the first block, it is used to store other, less sensitive data, related to the file. (Note that this limitation is easily overcome by extending the create file function to return the sector number of the first sector in the file.) For reading, the next sector to read is always available before the actual data is read, allowing the application to calculate the proper decryption key just in time.

The basic idea for this copy protection system is that de data is encrypted using a key that critically depends on the location in which the data is stored, which is combined by a method which renders it impossible to predict where data will be actually stored. Accordingly, copying of the data will result in a change of the storage location, breaking the critical relation between the latter and the encryption key. Thus, once the data has been moved, it can never be retrieved (assuming that the cryptography is sufficiently strong, the random number generator is cryptographically strong, and the shared secret is kept well hidden).

Digital Rights Management for Solid State Audio players

One of the short term goals of the Secure Digital Music Initiative (SDMI) is to specify a Digital Rights Management (DRM) system for portable audio players. This DRM system not only should enforce observance of copyright rules, but in addition should enable new business models as well (e.g. music rental, "try-before-you-buy," and controlled copying). By June 30, 1999, the specification should attain such a level as to enable development of the next generation of Solid State Audio (SSA) players.

Two basic tools are available to create a DRM system: encryption and watermarking. The former is used to control access to the audio, while the latter can be used to detect illegal copies which have been made using analogue reproduction means. To enable new business models as required by the Music Industry, a method should be provided to securely associate usage information with the audio content. On each access to the content, the usage information should be updated and checked against the rights that were purchased, like a ticket. Of course, it is required to prevent (or detect) so called "replay attacks," in which the updated audio content is replaced by the originally purchased pristine (i.e. unused) audio content, which could have been stored in an archive (e.g. on a PC hard disk).

In the following, it is assumed that the SSA player employs detachable memory modules, which can be accessed by non-compliant devices as well. For consumer convenience, it should be possible to plug a loaded memory module in any compliant SSA player and have the audio play, without the need for an on-line transaction, or for two players to be physically connected. After reviewing some simple copy protection approaches, a method is proposed which allows prevention of replay attacks under these conditions.

A very simple and straightforward approach to copy protection for SSA is to embed a unique identification code in each memory module, which can not be modified by a user. Alternatively, a "fingerprint" of each memory module may be obtained from the location of bad (defect) blocks in the device, and used as the identification code. This code can be combined with a secret key, which is shared by all or a group of SSA players, to yield an encryption key which is specific for that memory module. Since the secret key is shared, the module specific key can be recovered in each SSA player in the group, allowing the audio content to be moved from one player to another.

However, this approach does not protect against replay attacks, which can be seen as follows. Once the audio content is downloaded in the module, it can be read off the module using a non-compliant device, and stored in an archive. The audio can not be played from the archive because it is stored in encrypted form. However, as soon as the content on the memory module has been expired, it can be replaced by a fresh copy from the archive. It is clear that this can be repeated indefinitely, and as such this method is not suitable to implement new business models such as music rental etc.

One solution to this problem is to equip the memory module with a smart card IC, which controls access to the memory using some authentication protocol (e.g. based on public key cryptography). This would prevent a non-compliant device to copy the module content to an archive, and subsequently to restore it after the original on the module has been expired. However, this is a quite costly solution. Moreover, due to the limited processing power of such an IC, an authentication protocol based on public key cryptography may result in an unacceptable delay before the audio starts playing.

Here, an alternative approach is proposed, which is expected to be much cheaper, and which does not require a lengthy authentication procedure. The basic idea is that to prevent a replay attack, it is required to design a module which enters different states even though the data being stored is identical. The method as presented below using the example of a solid state memory module is applicable as well to other kinds of storage modules that have some (simple) control logic built-in. Finally, the method provides a flexible framework by means of which a copy protection system can be built, without imposing limitations on the usage of the module.

Current flash memory modules for solid state hard disk and multimedia storage applications are organised in 512 byte sectors. Associated with each sector is a tag area (usually 16 bytes), which is employed to store e.g. a bad block flag, a usage count (to implement wear levelling), and error correction information. This data typically is maintained by an on-board controller or the application. It is proposed to extend (or use part of) the tag area of each sector with a so-called "Secure Solid State Sector Tag," or S4T for short. This is field contains a random number, which has the following two basic properties:

- it is changed on each write access by some special (preferably on-chip) logic;
- it is read-only for external access.

Figure 1 shows the architecture of the proposed memory module. Ultimately, the S4T field should be integrated in the memory chip. However, as a near term (but much more expensive) solution, a memory chip can be placed on a module along with a controller. In that case, the latter is responsible for random number generation and implementation of the S4T field, e.g. by reserving part of the tag area for storage of the random number and denying write access to that part. Alternatively, the controller could reserve a few sectors for storage of the S4T fields, or provide a memory of its own for that purpose. Figure 2 gives some examples.

Next, it is detailed how the S4T field can be exploited to devise a copy protection system, which is resistant against replay attacks and thus enables new business models to be implemented. The audio content will be stored on the memory module in an encrypted form, either using a single key, or using a set of different keys in the case of block wise encryption. The rights which have been purchased with the content, and the usage information will also be stored on the module. These need not necessarily be encrypted, as will be shown later. Finally, the key(s) used to encrypt the content are stored on the module, encrypted with a key that is derived from

- the value of the S4T fields of the sectors in which the rights and usage information is stored;
- a secret key which is shared by all or a group of players;
- and optionally the value of some or all of the values in the S4T fields of the sectors in which the content is stored.

Figure 3 shows this in schematic form.

To analyse this copy protection system, it is assumed that a non-compliant device is used to read from and write to the memory module. Clearly, an exact bit copy of the audio content can be made to an intermediate storage device (e.g. a PC hard disk) since there are no restrictions whatsoever to reading the memory. However, this copy is unusable because it is encrypted with a key that can not be obtained (other than by reverse engineering the player to determine the shared secret key). On each play back of the audio content, the usage information is updated and checked against the rights. (Note that the key(s) used to encrypt the audio content are temporarily in the clear for play back, but well kept inside the player.) If the content has not been expired, the updated information is stored in the memory, and the key(s) used to encrypt the content are re-encrypted using the new value of the sectors in which the rights and usage information is stored. Now suppose that the audio content has been expired, and the bit copy has been placed back in memory. As it turns out, the result is not an exact bit copy, because the values in the S4T fields have been changed randomly on each write access to the memory. Therefore, the player will fail to recover the key(s) used to encrypt the audio content, since this requires the original values of the S4T fields (which are on the intermediate storage device, but can not be placed back in the memory module). Accordingly, the replay attack fails.

A further potential attack is to change the rights and usage information, which may have been stored in the clear. Again, the value of the S4T field of the sectors in which this information is stored will be irrevocably change, thus rendering recovery of the key(s) used to encrypt the content impossible. Thus, the attack fails (even if the rights and usage information is stored in the clear).

To defy this copy protection mechanism, an attacker has to build an interface which sits between the player and the memory module. The function of that interface is to intercept the values read from the S4T fields, and to replace them by some standard constant value. (Note that such an attack would work as well against copy protection mechanisms based on the use of a simple serial number on each memory module.) To avoid this attack, a costly mutual authentication of the memory module and player using smart card techniques is thought to be inevitable. However, it is surmised that this kind of attack is impractical in a portable device.

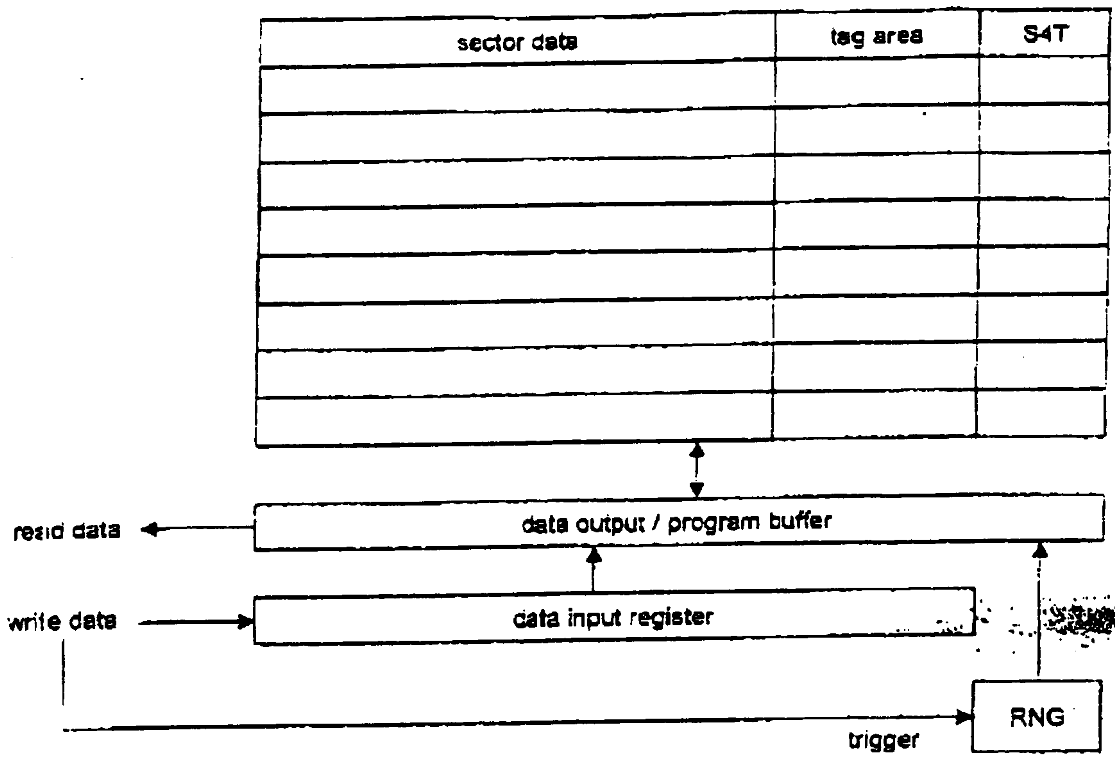

Figure 1. Schematic logical architecture of a memory module containing the proposed S4T field. On writing data to a sector in the memory, first the user data is stored in the data input register. Then as writing is triggered, the data is transferred to the program buffer, and the random number generator generates a new S4T value which is to be stored with the data. On reading data from a sector, all data is transferred to the data output buffer, including the S4T value. It is up to the application to make suitable use of the supplied information.

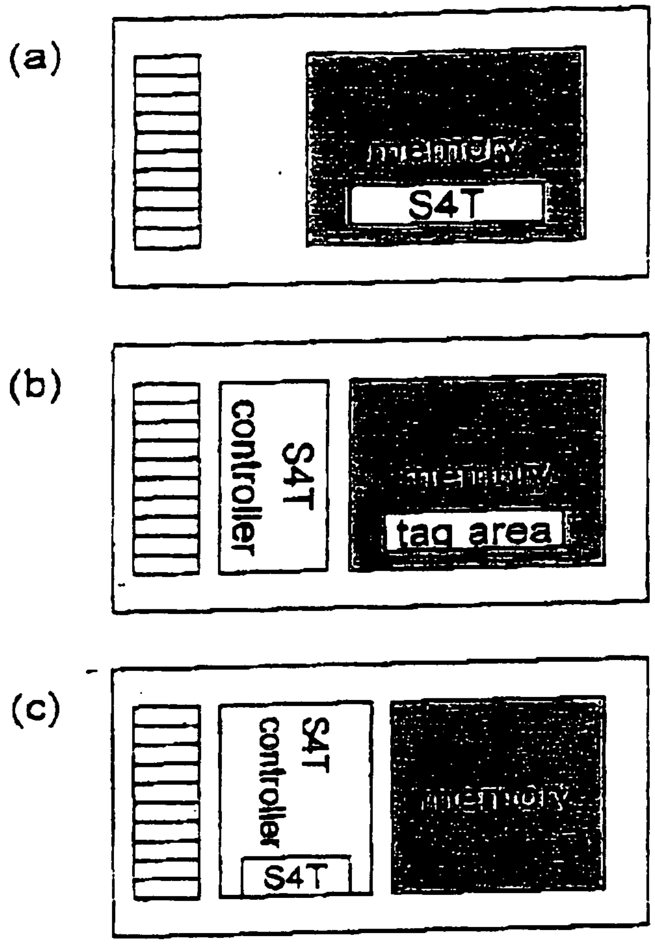

Figure 2. Some examples of memory modules with the proposed S4T functionality. (a) S4T functionality is integrated in the memory chip. (b) An existing memory chip is used in conjunction with a separate controller to implement the S4T functionality. Part of the tag area offered by the memory chip is employed to store the S4T data. (c) the same as (b), but now the controller has on-board storage space for the S4T data.

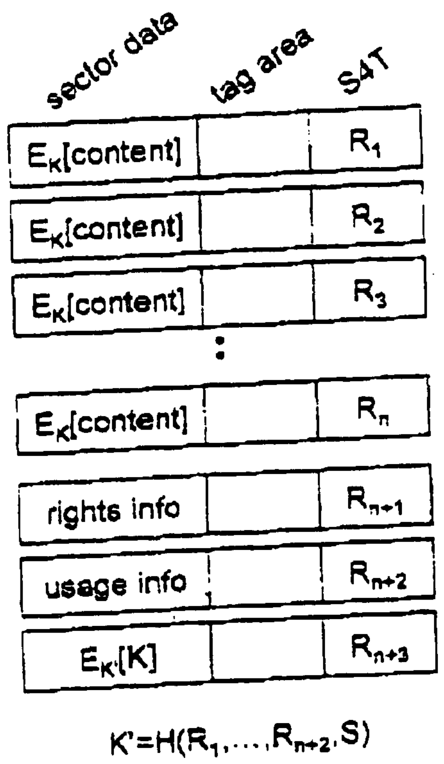

Figure 3. An example of the use of the S4T value to implement a copy protection method that is resistant to replay attacks. It is assumed that the content is encrypted with a key K, which is to be kept secret. Attached to the content is rights and usage information (the latter is updated on each access to the content). The encryption key K itself is stored after encryption with a key K', which is the output of a hash function H taking the S4T values $R_1,\ldots,R_{n+2}$ and the shared secret S as arguments. Note that K need not be a single value, but may be a set of keys as well.

Claims:

1. A method of copy protection substantially as described herein with reference to one or more of the accompanying drawings.

2. A method of copy protection in which a memory module which provides a tag associated with the stored data is used.

3. A method of copy protection in which a key which critically depends on the exact locations on which it is stored on the memory is used.

4. A device substantially as described herein with reference to one or more of the accompanying drawings.

5. A device wherein a method as claimed in Claim 1,2 or 3 is used for copy protecting the content stored in the device.

THIS PAGE BLANK (USPTO)